CS 2002 W14 Practical
180000560
14th May 2020

Overview

In the practical, first objective is to implement a fixed size generic stack. Then is to extend the implementation of thread-safe blocking stack.

Stack

Implementation and Design

Generic list is considered to implement the stack. By setting a top list node inside the generic stack, by switching the top list node during popping a new element it can satisfy the first in last out rules which is the basic property of generic stack. The new pushed element will be closer to the top list node. Pop action will read element from the list node from top list node, then go through the whole stack by using pointer of the list node.

Using generic list could save more memory allocation than fixed-size array. Memory allocated for fixed array will be occupied through whole stack operation even though some memory is not filled. A new list node's memory is allocated just after a new element being pushed, and it is also deallocated after an element is popped. All memory allocation is used only for the element that is currently in stack. Otherwise, it can be implemented to be not fixed-sized. The list node can be created and point to the next node without the reallocate the memory allocation for the whole stack implementation.

Test

Pop push and first in last out is essential features of generic stack. Most of the tests is implemented to test those features.

Pointer is allocated for testing. In each memory allocation, system will return a new unique pointer for each element. Therefore, pointer must point to the element we stored, only need to check the equality of pointer for testing.

As the pointer allocated each time is unique, it is simple to check if the stack following the rule of first in last out by comparing the pointer value. In this original stack, it can not accept more pushing operation or popping operation when size of stack reaches its limit. Some error message need to receive for indicating the user the condition of the stack.

Blockingstack

Implementation and Desgin

Most of the features in Blockingstack is similar to original stack. Some thread safe function needs to be considered when there are some variables that may be affected during multi-thread operations. And Blockingstack two operation: pop and push will wait for any available space instead of returning error message back to client.

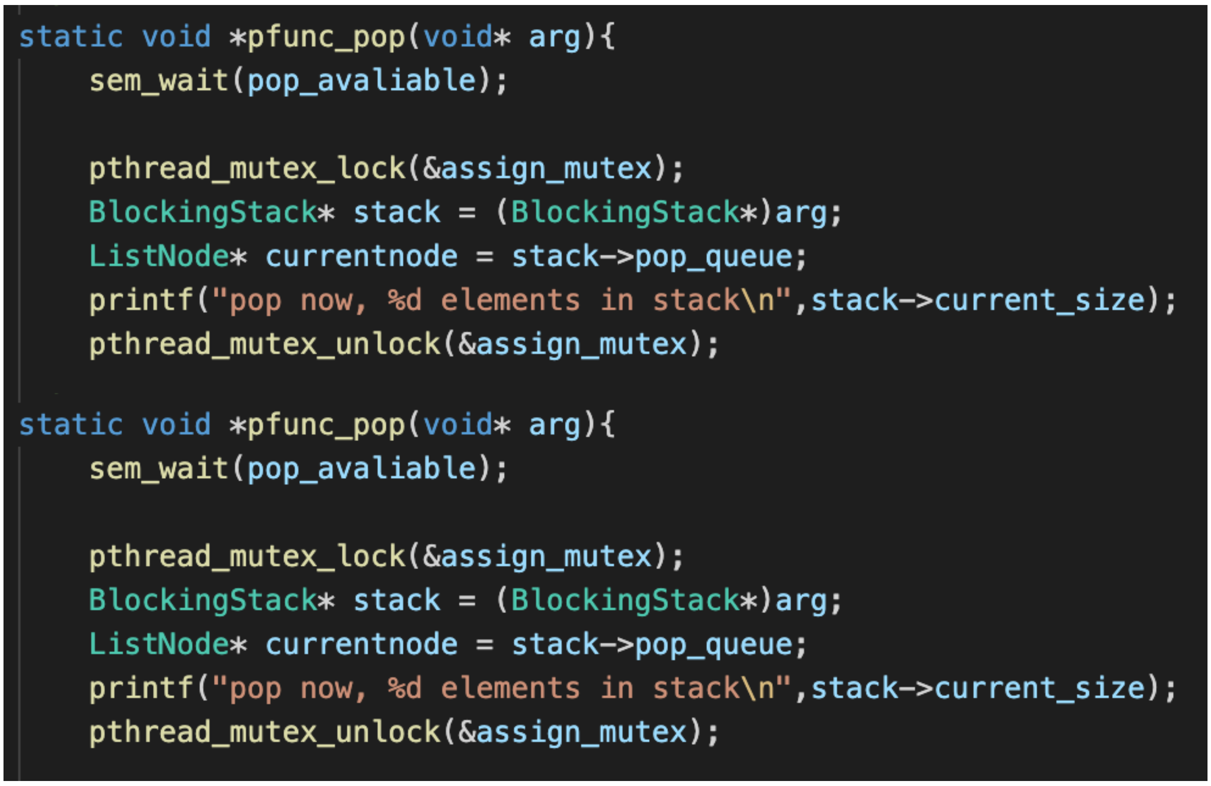
Two type of threads is considered, pop and push. Some implementations are made to adapt some features of Blockingstack. In Blockingstack all push operations will return true flag and create a new thread for push except null element. In default, all element is believed to be pushed into stack finally. New value is assigned for struct Blockingstack, the pointer to the top of the pop queues, it is used to store all popped elements. In phtread_join function, two thread will execute simultaneously, the parameter cannot return to back to main routine. Instead of returning the pointer of element, the pointer is stored into another list declared inside structure Blockingstack, all popped element will be released after two threads end its executions.

Test

Many features of BlockingStack is similar to stack, it is unnecessary to carry out repeated testing. Busy wating is one of the functions that need to be considered. Blockingstack itself is a fixed-sized generic stack, it can only hold limit number of elements. Two conditions will halt the threads: 1, There is no more free space for stack to push 2, There is no more elements for stack to pop. Therefore, to purpose of testing is to create threads that exceed the limit of stack. Then checking the threads of pop and push, push threads is executed when there is available space for pushing, so does pop thread.

Two ways is implemented to test, one is use testing function, which push and pop elements that have more number than maximum size of the Blockingstack which is included in TestBlockingStack. Another one is to use the print function and counter to check each thread if they are carried out the pop and push thread function in appropriate condition.

In pop and push thread, some print function will indicate each thread action by printing the counter number and thread name. It is carried out during testing environment.

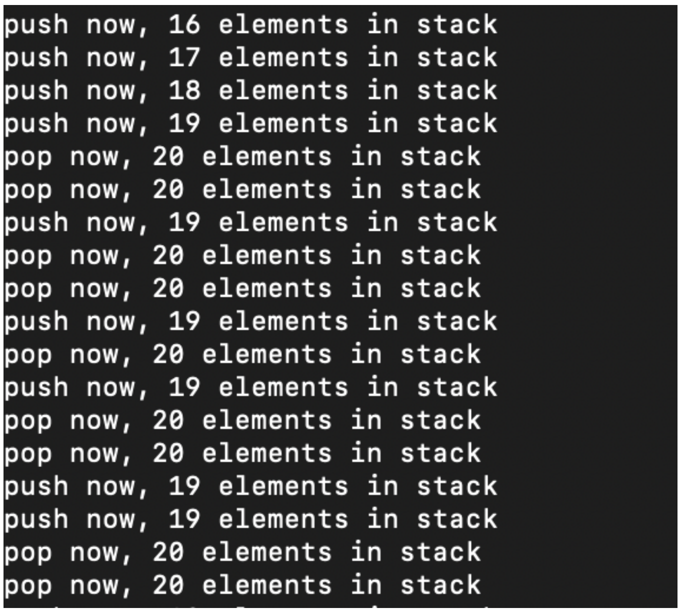


```
static void *pfunc_pop(void* arg){
    sem_wait(pop_avaliable);

    pthread_mutex_lock(&assign_mutex);
    BlockingStack* stack = (BlockingStack*)arg;
    ListNode* currentnode = stack->pop_queue;
    printf("pop now, %d elements in stack\n",stack->current_size);
    pthread_mutex_unlock(&assign_mutex);

static void *pfunc_pop(void* arg){
    sem_wait(pop_avaliable);

    pthread_mutex_lock(&assign_mutex);
    BlockingStack* stack = (BlockingStack*)arg;
    ListNode* currentnode = stack->pop_queue;
    printf("pop now, %d elements in stack\n",stack->current_size);
    pthread_mutex_unlock(&assign_mutex);
```



```
push now, 16 elements in stack
push now, 17 elements in stack
push now, 18 elements in stack
push now, 19 elements in stack
pop now, 20 elements in stack
pop now, 20 elements in stack
push now, 19 elements in stack
pop now, 20 elements in stack
pop now, 20 elements in stack
push now, 19 elements in stack
pop now, 20 elements in stack
push now, 19 elements in stack
pop now, 20 elements in stack
pop now, 20 elements in stack
push now, 19 elements in stack
push now, 19 elements in stack
pop now, 20 elements in stack
pop now, 20 elements in stack
```

By the printing function inside thread, pop and push threads execute when there is space available, the default maximum size of stack is 20, therefore any thread will wait when they meet the boundary.

Evaluation

Thread safety is quite important when different threads acquired for the same resources, pop and push thread both need to access the stack, any missing mutex lock will cause undefined behaviour after the thread execution. Deadlock is quite common when handling waiting function. There are some functions that the Blockingstack can not satisfy, if the number of threads is not equal between pop and push. Deadlock will happen, and all function have to be stalled in waiting status. Otherwise, to create multiple pop and push action without considering order. The only best solution I could think of is to create the two thread and use pthread_join to handle the multiple thread with semaphores and mutex functions. It is hard to return parameter from the threads; better way is to use pointer for storing the parameter that need to be passed by threads. There are some small difference between implemented code and description on the header file.

Conclusion

Original stack could store fixed size of any elements. Blockingstack enhance stack performance on thread safety and blocking function. Both extended and original function could be used in different condition as a generic stack.